

PREEMPTIVE EVICTION OF STALE ENTRIES IN A COMPUTER CACHE BY USE OF AGE-BITS

FIELD OF INVENTION

5

This invention relates generally to computer systems and more specifically to cache memory systems.

BACKGROUND OF THE INVENTION

10

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
220

All processors and caches may share a common main memory. A particular line may simultaneously exist in memory and in the cache hierarchies for multiple processors. All copies of a line in the caches must be identical, a property called coherency. The protocols for maintaining coherence for multiple processors are called cache coherence protocols.

Cache coherence protocols commonly place each cached line into one of multiple states. One common approach uses three possible states for each line in a cache. Before any lines are placed into the cache, all entries are at a default state called "Invalid". When a previously uncached physical line is placed into the cache, the state of the entry in the cache is changed from Invalid to "Shared". If a line is modified in a cache, it may also be immediately modified in memory (called write through). Alternatively, a cache may write a modified line to memory only when the modified line in the cache is invalidated or replaced (called write back). For a write-back cache, when a line in the cache is modified, or will be modified, the state of the entry in the cache is changed to "Modified". The three-state assignment just described is sometimes called a MSI protocol, referring to the first letter of each of the three states.

To improve performance, the computer system tries to keep data that will be used soon in the fastest memory, which is usually a cache high in the hierarchy. Typically, when a processor requests a line, if the line is not in a cache for the processor (cache miss), then the line is copied from main memory, or from a cache of another processor. A line from main memory, or a line from another processor's cache, is also typically copied into a cache for the requesting processor, assuming that the line will need to be accessed again soon. If a cache is full, then a new line must replace some existing line in the cache. If a line to be replaced is clean (the copy in cache is identical to the copy in main memory), it may be simply overwritten. If a line to be replaced is dirty (the copy in cache is different than the copy in main memory), then the line must be evicted (copied to main memory). A

replacement algorithm is used to determine which line in the cache is replaced. A common replacement algorithm is to replace the least-recently-used line in the cache.

One particular performance concern for large multiple processor systems is the impact on latency when one processor requests a line that is cached by another processor. If a modified (dirty) line is cached by a first processor, and the line is requested by a second processor, the line is written to main memory, and is also transferred to the requesting cache (called a cache-to-cache transfer). For a large multiple-processor system, a cache-to-cache transfer may require a longer latency than a transfer from main memory. In addition, for a large multiple-processor system, a cache-to-cache transfer may generate traffic on local buses that would not be required for a transfer from main memory. Accordingly, average latency can be improved by reducing the number of cache-to-cache transfers, which in turn can be improved by preemptive eviction of stale dirty lines.

In addition, for large systems, even if a line in another processor's cache is not dirty, there may be substantial latency involved in determining whether the line is actually dirty. For example, in the MSI coherency protocol, if a line is in the Modified state, one processor may modify the line without informing any other processor. A line in the Modified state a cache may actually be clean, which means that the copy in main memory may be used, but a substantial time may be required to determine whether the line is clean or dirty. Therefore, average latency may be improved by preemptive eviction of stale lines in the Modified state, even if they are clean.

Systems for determining the age of dirty lines are known. For example, U.S. Patent Number 6,134,634 describes a system in which each line in a cache has an associated counter that is used to count cycles during which the line has not been written. If the count exceeds a predetermined number, the line is determined to be stale and may be evicted. There is a need for lower cost identification of stale lines.

SUMMARY OF THE INVENTION

5 In an example embodiment of the invention, a cache system can identify stale lines, with very little incremental circuitry. In particular, the example cache system can determine that a predetermined time has elapsed since a line of the cache has been accessed (or alternatively, modified), with only a single bit per line instead of a multiple bit counter or timer per line. A single age-bit may provided for each line in the cache, or a single age-bit may be provided for each index. The age-bits are initially set to a first logical state. Each time a line, or index, is accessed (or alternatively, written), by a processor, the corresponding age-bit is reset to the first logical state. A state machine periodically checks the status of each age-bit. If the state machine detects that an age-bit is in the first logical state, the state machine sets the age-bit to a second logical state. If the state machine detects that an age-bit is already in the second logical state, then the set of lines corresponding to the index corresponding to the age-bit, or line of data corresponding to the age-bit, has not been accessed or changed since the last time the state machine checked the age-bit, and is therefore, stale. If there is one age-bit per line, the line may be pre-emptively evicted. If there is one age-bit per index value, a replacement algorithm may be used to determine a line to evict, for example, the least-recently-used line
10
15
20 corresponding to the index.

BRIEF DESCRIPTION OF THE DRAWINGS

25 Figure 1 is a block diagram of an example system that includes the invention.

Figures 2A and 2B are flow charts of event-driven methods in accordance with part of an example embodiment of the invention.

Figure 3 is a flow chart of a method in accordance with part of an example embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT OF THE INVENTION

If a cache stores an entire line address along with the data, and any line can be placed anywhere in the cache, the cache is said to be fully associative. However, for a large cache in which any line can be placed anywhere, the hardware required to rapidly determine if an entry is in the cache (and where) may be very large and expensive. For large caches, a faster, space saving alternative is to use a subset of an address (called an index) to designate a set of lines within the cache, and then store the remaining set of more significant bits of each physical address (called a tag) along with the data. In a cache with indexing, an item with a particular address can be placed only within a set of lines designated by the index. If the cache is arranged so that the index for a given address maps to exactly one line in the subset, the cache is said to be direct mapped. If the index maps to more than one line in the subset, the cache is said to be set-associative. All or part of an address is hashed to provide a set index which partitions the address space into sets.

In many computer memory architectures, a processor produces virtual addresses that are translated by a combination of hardware and software to physical addresses, which access physical main memory. A group of virtual addresses may be dynamically assigned to each page. Virtual memory (paging or segmentation) requires a data structure, sometimes called a page table, that translates the virtual address to the physical address. To reduce address translation time, computers commonly use a specialized associative cache dedicated to address translation, commonly called a Translation Look-aside Buffer (TLB).

Figure 1 illustrates an example cache system in which the invention may be implemented. The specific example cache illustrated in figure 1 is a four-way set-associative cache with virtual addressing. However, the invention is applicable to any cache configuration, including direct mapped caches, fully associative caches, or other configurations of set associative caches. In the example of figure 1, a virtual address 100 comprises lower order index bits 102 and upper order tag bits 104. The index bits are typically the same for the virtual address and the physical address. The index bits are used to select one set of lines of data in the data section 106 of the cache. The output of data section 106 is four lines of data. The index bits are also used to select a set of physical tags in a tag section 108 of the cache. The output of the tag section 108 is four physical tags, each corresponding to one data line. The TLB 110 stores both virtual and physical tags. For a TLB hit, the TLB provides a physical tag that corresponds to the virtual tag 104. Each of four digital comparators (not illustrated) then compares the physical tag from the TLB to a physical tag from the tag section 108. A matching pair of physical tags indicates through logic (not illustrated) which one of four lines of data is selected by a multiplexer (not illustrated). Note that for the particular index bits there may not be a matching pair of physical tags, in which case there is a cache miss. A fully associative cache may have a structure similar to a TLB to determine whether a line is in the cache, and where.

In accordance with the invention, age-bits are used to indicate whether lines in the cache may be stale. An age-bit is associated with each possible index value, or alternatively with each line in the cache. In figure 1, box 112 indicates a set of age-bits, with each age-bit associated with one index value. Boxes 114 indicate an alternative design, with four sets of age-bits, with each age-bit associated with one line of data. For a fully associative cache, each age-bit would be associated with one line of data. In a direct mapped cache, each age-bit would be associated with both one index value and with one line of data. It is not necessary for the age-bits to

be physically located in the tag section 108 of the cache. It is only necessary to have a one-to-one relationship with index values, or alternatively with data lines. It is common to store cache coherency information in the tag section 108, so that age-bits can be added to the tag section with little incremental hardware. In addition, for some processor architectures, the tag section 108 may be integrated onto the processor chip, and the data structure 106 may be on a separate chip. If the tag structure 108 is an integral part of the processor, access times for age-bits is decreased, facilitating manipulation of age-bits during the latency for data retrieval. For a fully associative cache, the age-bits may be physically located with the TLB-like structure used to determine whether a line of data is in the cache.

A state machine 116 periodically cycles through all the age-bits, as discussed in more detail in conjunction with figure 3. For purposes of illustration only, in figure 1, the state machine 116 is shown as interacting with age-bits 112 associated with each index value. If the age-bits are associated with every line of the data (age-bits 114), then the state machine would interact with age-bits 114.

Figures 2A and 2B illustrate example alternative event driven methods, and figure 3 illustrates an example method for the state machine (figure 1, 116). All the age-bits are initially preset to a first logical state (for example, logical ZERO) (figure 3, 300). The age-bits may then be used to detect whether a line has been accessed (read or write) (figure 2A), or alternatively whether a line has been modified (write only) (figure 2B). If the goal is to identify stale lines, then each time a line is accessed (figure 2A, 200), the corresponding age-bit is set to the first logical state (figure 2A, 202). Alternatively, if the goal is to identify stale dirty lines, each time a line is modified (written) (figure 2B, 204), the corresponding age-bit is set to the first logical state (figure 2B, 206). If the age-bits are physically part of the cache, they may be set to the first logical state by the cache. If the age-bits are physically separate from the cache, the state machine (figure 1, 116) may

receive the index value and the state machine may set the corresponding age-bit to the first logical state.

Figure 3 illustrates an example method for the state machine (figure 1, 116), assuming that there is one age-bit for each index value (figure 1, 112). At step 300, each age-bit is initialized to a first logical state, for example, logical ZERO. The index is initialized at step 302. The state machine then waits for a predetermined amount of time (step 304) before checking the status of the age-bits. The state machine then cycles repeatedly through all index values. At step 306, for each index value, the state machine examines the state of the corresponding age-bit. If the age-bit is in the first logical state, then at step 308, the state machine sets the age-bit to a second logical state (for example, logical ONE).

If the age-bit is already in the second logical state, at step 310, one line in the set of lines corresponding to the index value may be evicted. In a set associative cache, there are multiple lines that correspond to the index value, and the system must determine which of the multiple lines to evict. There may be more than one stale line corresponding to the index value. It is common for caches to have a replacement algorithm, for example, least-recently-used. For the example of a four-way set-associative cache, the replacement algorithm may be used to select one of four lines associated with an index value having a corresponding age-bit in the second logical state. There are several alternatives. If the goal is to detect and evict stale lines, then the replacement algorithm may be used to select any of the lines corresponding to the index value. In particular, if the replacement algorithm is least-recently-used, a stale line will be evicted. If the goal is to detect stale dirty lines, then the replacement algorithm may be limited to just the modified lines corresponding to the index value.

Steps 312 and 314 cycle steps 306-310 through all the index values, and then execute the wait period (step 304) before repeating the cycle.

In the example embodiment of figure 3, the wait time 304 is a minimum wait time, and the overall cycle time for checking status for any particular age-bit may be longer than the minimum wait time by a variable amount. For example, it may be preferably to execute the index loop (figure 3, steps 306-314) only when the cache is otherwise idle, so the total cycle time for checking the status for any one age-bit may depend on how busy the cache is. In addition, recall that each time a line is accessed (or alternatively, each time a line is written), the corresponding age-bit is set to the first logical state. As a result, setting age-bits to the first logical state (figures 2A and 2B) is asynchronous to the method of figure 3. Accordingly, the time between when an age-bit is set to the first logical state, and the time at which it is checked by the state machine, is variable. As an alternative, setting age-bits to the first logical state could be made synchronous, for example by delaying setting age-bits until just before or after step 304 of figure 3, and the state machine could be implemented with non-variable cycle times.

If there is one age-bit per line (figure 1, 114), for the example of a four-way set-associative cache, then the system using the example method of figure 3 could repeat steps 306 through 310 four times, once for each of four age-bits associated with each index value. For a fully associative cache, the system could cycle through the entries in a look-up structure (for example, a content-addressable-memory) instead of index values.

In the resulting cache system, stale lines, or alternatively stale dirty lines, can be identified by adding only one bit for each of the number of index values in the cache, or one bit for each of the number of lines in the cache, plus a state machine.

The foregoing description of the present invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed, and other modifications and variations may be possible in light of the above teachings. The embodiment was

chosen and described in order to best explain the principles of the invention and its practical application to thereby enable others skilled in the art to best utilize the invention in various embodiments and various modifications as are suited to the particular use contemplated. It is intended that the appended claims be construed to include other alternative embodiments of the invention except insofar as limited by the prior art.

5

HP CASE # 10013442